Project Explanation: PAQJP_6_Smart Compression System with Dictionary

The **PAQJP_6_Smart Compression System with Dictionary** is a Python-based lossless compression program that integrates two distinct compression algorithms: the **Smart Compressor**, which uses dictionary-based SHA-256 hash verification, and the **PAQJP_6 Compressor**, which applies a series of reversible transformations optimized for different file types. The system evaluates both methods for each input file and selects the one producing the smallest output, prepending a marker (`00` for Smart Compressor, `01` for PAQJP_6) to indicate the method used. Designed for efficient compression of text, JPEG, and other files, it includes special handling for dictionary files (e.g., `.paq` files) to output compact 8-byte SHA-256 hashes when applicable, eliminating the need for full compression in those cases.

Core Functionality

The project combines two compression approaches to maximize efficiency while ensuring lossless data recovery:

1. **Smart Compressor**:

   - **Dictionary-Based Hash Verification**: Computes the SHA-256 hash of the input file and searches for it in a predefined set of dictionary files (e.g., `eng_news_2005_1M-sentences.txt`, `words.txt.paq`). If the hash is found, it logs the match, indicating the file is known.

   - **Special Handling for `.paq` Files**: For files named `words.txt.paq`, `lines.txt.paq`, or `sentence.txt.paq`, if the file size exceeds 8 bytes, the system outputs only an 8-byte SHA-256 hash, significantly reducing storage for known files.

   - **Compression Process**: For other files, applies a reversible XOR transform (using 0xAA), compresses the data with the PAQ9a algorithm (via the `paq` module), and includes a 32-byte SHA-256 hash for integrity verification. The output format is `[0x00][32-byte SHA-256 hash][compressed data]` if compression is efficient; otherwise, it skips compression.

   - **Decompression**: Reverses the process, verifying the stored SHA-256 hash against the decompressed data to ensure accuracy.

2. **PAQJP_6 Compressor**:

   - **Transformations**: Applies a series of reversible transformations (11 predefined and up to 244 dynamic ones) to preprocess data for better compression. Transformations include

XOR operations with prime numbers, pi digits, or file-size-based values, tailored to improve compressibility.

- **Compression Modes**: Offers fast mode (transformations 1, 3, 4–9) and slow mode (transformations 1–11, 12–255). Text (`.txt`) and JPEG (`.jpg`, `.jpeg`) files prioritize transformations 7–9 (and 10–11 in slow mode) for better performance.

- **Compression Methods**: Uses PAQ9a for most files and Huffman coding for files smaller than 1024 bytes (marked with transform marker 4). The output format is `[0x01][1-byte transform marker][compressed data]`.

- **Decompression**: Reads the transform marker to apply the appropriate reverse transformation after PAQ9a or Huffman decompression.

3. **Combined Compression**:

- The system runs both Smart Compressor and PAQJP_6 on the input file, comparing the output sizes.

- It selects the smaller output, prepending `00` or `01` to indicate the method.

- For empty files, it outputs a single byte `[0]`. For `.paq` dictionary files, the Smart Compressor's 8-byte hash output often results in the smallest size.

Key Features

- **Dictionary-Based Optimization**: The Smart Compressor leverages a list of dictionary files (`DICTIONARY_FILES`) to check if the input file's SHA-256 hash exists, enabling compact storage for known files. For `.paq` files, it outputs an 8-byte hash if the file is larger than 8 bytes, achieving significant space savings.

- **Dual Compression Strategy**: By testing both algorithms, the system ensures the smallest possible output, balancing the Smart Compressor's hash-based efficiency with PAQJP_6's transformation flexibility.

- **File Type Optimization**: Uses a `Filetype` enum (`DEFAULT = 0`, `JPEG = 1`, `TEXT = 3`) to detect file extensions and prioritize transformations for text and JPEG files, improving compression ratios.

- **Lossless Integrity**: The Smart Compressor includes a 32-byte SHA-256 hash in its output for verification during decompression, ensuring no data loss. PAQJP_6 uses reversible transformations to guarantee lossless recovery.

- **No Datetime Dependency**: The system excludes datetime encoding, simplifying the output format and focusing solely on data compression.

- **Flexible Modes**: Supports fast mode for quicker compression with fewer transformations and slow mode for maximum compression with all transformations.

- **Robust Error Handling**: Includes comprehensive logging for file I/O, compression failures, dictionary loading, and hash verification, ensuring reliable operation even with missing or invalid inputs.

Technical Components

- **Dictionary Management**:

  - Loads text content from files listed in `DICTIONARY_FILES`.

  - Searches for SHA-256 hashes in these files to identify known inputs.

  - Handles missing or unreadable dictionary files with warnings, allowing the program to continue.

- **Smart Compressor**:

  - Computes SHA-256 hashes (hexadecimal for searching, binary for output).

  - Applies a simple XOR transform (0xAA) before PAQ9a compression.

  - Outputs compact 8-byte hashes for `.paq` dictionary files when applicable.

- **PAQJP_6 Compressor**:

  - Implements transformations like XOR with primes, pi digits, or dynamic patterns.

  - Uses Huffman coding for small files and PAQ9a for larger ones.

  - Dynamically generates transformations 12–255 for extensive testing in slow mode.

- **Pi Digits**: Uses 3 mapped pi digits (e.g., `[3, 1, 4]` if `mpmath` is unavailable) for transformations 7–9, ensuring consistent behavior.

- **File Type Detection**: Identifies `.txt`, `.jpg`, and `.jpeg` files to optimize transformation selection, defaulting to `DEFAULT` for others.

- **Output Formats**:

  - Smart Compressor: `[0x00][32-byte SHA-256 hash][PAQ9a-compressed data]` or `[0x00][8-byte SHA-256 hash]` for `.paq` files.

  - PAQJP_6: `[0x01][1-byte transform marker][PAQ9a or Huffman compressed data]`.

Purpose and Use Cases

The system is designed for applications requiring high compression ratios and data integrity, such as:

- **Data Archival**: Efficiently stores large datasets, especially when dictionary files contain hashes of common files.

- **Backup Systems**: Reduces storage requirements for backups with lossless recovery.

- **Research**: Serves as a platform for experimenting with compression algorithms, dictionary-based techniques, and transformation strategies.

- **Specialized File Handling**: Optimizes storage for specific files (e.g., `.paq` dictionary files) by using compact hash representations.

Limitations

- **Performance**: PAQ9a compression is computationally intensive, particularly in slow mode with many transformations.

- **Dictionary Dependency**: Effectiveness for `.paq` files relies on the presence and accuracy of dictionary files.

- **File Type Support**: Limited to text and JPEG optimization; other file types use default settings.

- **Huffman Coding**: Only applied to files < 1024 bytes, which may not always be optimal.

This project provides a powerful, flexible compression system that balances dictionary-based efficiency with transformation-based optimization, making it a valuable tool for advanced compression tasks.